Disclaimer: Vsaka podobnost z resničnimi osebami, vojskami, podjetji, ministrstvi ali sodišči je večinoma naključna.

Nadaljujemo, kjer smo ostali prejšnjič.

Tokrat ni dodatne naloge.

Tole je ena velika blamaža. V treh točkah.

1. Gospa Kozjančeva bi strateški plan napada na Marsovce hotela oddati v Excelu. *Centralni računalnik za koordinacijo napadov na vesoljce* (CRKnaV) seveda ne zna brati Excela. Gospa Kozjančeva je zato kot primer, kaj kani sproducirati, natipkala tole skrpucalo

   ```
   4.25, 8.5:   5.0, 5   |   8.125, -7.5    |   3.5, -6.75
   6.0, 1.25:  15.5, 5.5  |  1.5, -1.25 | 2.5, -9.5625  | -3, -3
   8, 1: -2, 5
   ```

   in v mail napisala, da "*so u vsaki vrstici koordinate naše ladje pol pa še marsovskih in tko bom jest delala sploh nej me pa Grabnar iz general štaba neha [cenzurirano], drgač bom pač dala odpovet grem pač rajš delat na kakšno normalno ministrstvo kjer majo excel k to znam*".

   Podatki bodo torej v takšni obliki, le številke bodo drugačne. Trenutno so še vojaška skrivnost, ki jo pozna samo gospa Kozjančeva.

2. Nadalje se je izkazalo, da CRKnaV pričakuje podatke v takšni obliki.

   ```
   1:      4.2500      8.5000 ->      5.0000      5.0000
                                      8.1250      8.1250
                                      3.5000      3.5000
   2:      6.0000      1.2500 ->     15.5000      5.5000
                                      1.5000      1.5000
                                      2.5000      2.5000
                                     -3.0000     -3.0000
   3:      8.0000      1.0000 ->     -2.0000      5.0000
   ```

   Z vsemi decimalkami in presledki in vsem. Kolikor je Kozjančeva šlampasta do enomoglosti, je ta picajzlast do skrajnosti. Vsaka številka ima morebitni `-`, nato do štiri mesta levo od pike, štiri za piko, vmes presledki, pa dvopičja in puščice, točno tako in nič drugače. Ker je vojska vojska. Čeprav slovenska.

3. Kot da ta zafrkancija s Kozjančevo in CRKnaV še ne bi bila zadosti, se je zgodilo naslednje. Neko podjetje je začelo izdelovati softver za pretvorbo, potem pa se je konkurenčno podjetje pritožilo na razpis, zadeva je šla do Ustavnega sodišča in le-to je razpis razveljavilo, saj škoda, ki bi nastala zaradi marsovskega napada ni sorazmezna s škodo, ki bi nastala s kršenjem zakonodaje s področja javnih naročil. Vse, kar je ostalo od njihovega dela, je nekaj testov. Znajti se bo potrebno iz njih.

Videti je, da bo potrebno napisati tri funkcije.

- `preberi(ime_datoteke)` dobi ime datoteke, kakršne proizvaja Kozjančeva in, kot je sklepati iz testov, vrne slovar, katerega ključi so koordinate naših ladij, pripadajoče vrednosti pa seznami koordinat marsovskih ladij. Ime datoteke je načelno poljubno.
- `vrstica(i, ladja, marsovec)` vrne eno vrstico navodil za CRKnaV. Iz testov lahko sklepamo, da `i` vsebuje zaporedno številko naše ladje ali pa `None`. Če je `i` številka, jo izpiše, nato izpiše koordinate naše in koordinate marsovske ladje. Če je `i` enak `None`, pa ne izpiše ne številke na koordinat naše ladje, temveč samo koordinato marsovske ladje. Torej: klic `vrstica(7, (4.25, 8.50), (5, 5))` vrne

  ```
  7:     4.2500     8.5000 ->     5.0000    12.3000
  ```

  klic `vrstica(None, (4.25, 8.50), (5, 12.3))` pa bi vrnil

  ```
                                  5.0000    12.3000
  ```

  O tem, kako, točno mora biti oblikovan izpis, bo potrebno sklepati iz testov. Good luck.
- `navodila(razpored, ime_datoteke)` dobi slovar v takšni obliki, kot ga vrača prva funkcija in ime datoteke, v katero izpiše navodila, ki se do zadnje pike ujemajo s predvidenim formatom.

Sreča v nesreči: v tej nalogi se se dogaja samo v dveh dimenzijah.

Rešitev

`preberi(ime_datoteke)`

Pripravimo prazen slovar. Gremo čez vrstice datoteke; videti bodo, na primer, tako: `"4.25, 8.5: 5.0, 5 | 8.125, -7.5 | 3.5, -6.75"`. Vsako razbijemo po `":"`; dobljena dela (vedno bosta dva) bomo poimenovali `ladja` in `marsovci`. Ladjo razbijemo po `","`, da dobimo koordinati, `x` in `y`. Marsovce razbijemo po `|`, da dobimo posamezne marsovske ladje; vsako od njih pa v notranji zanki potem še po `","`, da dobimo posamezni koordinati. Koordinate zlagamo v seznam koordinat. Na koncu ga dodamo v slovar, pod ključem, ki predstavlja koordinati naše ladje.

V spodnji kodi sem nekoliko pretiraval s praznimi vrsticami - to pa zaradi primerov, ki sledijo.

```
def preberi(ime_datoteke):
    razpored = {}

    for vrstica in open(ime_datoteke):
        ladja, marsovci = vrstica.split(":")
```

```
        x, y = ladja.split(",")

        koordinate = []
        for marsovec in marsovci.split("|"):
            mx, my = marsovec.split(",")
            koordinate.append((float(mx), float(my)))

        razpored[(float(x), float(y))] = koordinate

    return razpored

preberi("jozica-kozjanc.txt")
```

```
{(4.25, 8.5): [(5.0, 5.0), (8.125, -7.5), (3.5, -6.75)],
 (6.0, 1.25): [(15.5, 5.5), (1.5, -1.25), (2.5, -9.5625), (-3.0, -3.0)],
 (8.0, 1.0): [(-2.0, 5.0)]}
```

V študentskih rešitvah je pogosto videti tole:

```
def preberi(ime_datoteke):
    razpored = {}
    koordinate = []

    for vrstica in open(ime_datoteke):
        ladja, marsovci = vrstica.split(":")
        x, y = ladja.split(",")

        for marsovec in marsovci.split("|"):
            mx, my = marsovec.split(",")
            koordinate.append((float(mx), float(my)))

        razpored[(float(x), float(y))] = koordinate
        koordinate = []

    return razpored
```

Dasiravno to deluje,

```
preberi("jozica-kozjanc.txt")
```

```
{(4.25, 8.5): [(5.0, 5.0), (8.125, -7.5), (3.5, -6.75)],
 (6.0, 1.25): [(15.5, 5.5), (1.5, -1.25), (2.5, -9.5625), (-3.0, -3.0)],
 (8.0, 1.0): [(-2.0, 5.0)]}
```

mi ni všeč. Razlika je, kot vidite, v tem, da prazen seznam koordinat ustvari na začetku funkcije, pred zunanjo zanko, potem pa ga znotraj zunanje zanke pobriše. To zakrije resnično logiko programa. V gornji verziji imamo blokec

ki pripravi seznam in ga napolni. V spodnji različici imamo samo

seznam `koordinate` pa se v bistvu ustvarja precej prej in nekoliko kasneje.

Druga, bolj deviantna različica, ki sem jo videl, je tale.

```
def preberi(ime_datoteke):
    razpored = {}
    koordinate = []

    for vrstica in open(ime_datoteke):
        ladja, marsovci = vrstica.split(":")
        x, y = ladja.split(",")

        for marsovec in marsovci.split("|"):
            mx, my = marsovec.split(",")
            koordinate.append((float(mx), float(my)))

        razpored[(float(x), float(y))] = koordinate
        koordinate.clear()

    return razpored
```

Tole pa prav poučno ne deluje.

```
preberi("jozica-kozjanc.txt")
```

```
{(4.25, 8.5): [], (6.0, 1.25): [], (8.0, 1.0): []}
```

Ta funkcija ustvari en in en sam seznam `koordinate`, ki ga malo polni in prazni. Ker v slovar kot vrednost vstavlja ta, vedno en in isti seznam, se vse vrednosti nanašajo pač nanj in so med seboj enake. Še več, med seboj so *iste*. In to takšne, kot je ta seznam na koncu, torej prazen, saj je zadnja stvar, ki se mu zgodi, `koordinate.clear()`.

`vrstica(i, ladja, marsovec)`

Rešitev je takšna.

```
def vrstica(i, ladja, marsovec):
    if i is None:
        return f"{' ':4}  {' ':10} {' ':10}    {marsovec[0]:10.4f} {marsovec[1]:10.4f}"
    else:
        return f"{i:4}: {ladja[0]:10.4f} {ladja[1]:10.4f} -> {marsovec[0]:10.4f} {marsovec[1
```

Kaj bomo izpisali, je odvisno od `i`. Najprej poglejmo drugo različico, tisto, ko `i` ni `None`. Tam pač izpišemo, kar je treba in kakor je treba: `i` na štiri mesta, pa koordinati ladij na deset mes s štirimi decimalkami, pa puščico in koordinate marsovcev na prav tak način. Nič posebnega, samo mesta je potrebno prav prešteti. In nize je potrebno znati oblikovati.

Bolj zanimiva je prva različica - prav zato, ker je tako dolgočasna. Tu prvi del praktično prepišemo iz druge, le da `i` in koordinati ladje zamenjamo s presledki.

Seveda bi lahko preprosto prešteli presledke in jih izpisali, kolikor jih je treba. Vendar je bolj pregledno in varneje imitirati drugo vrstico.

Kdor res ne mara ponavljanja, pa napiše:

```
def vrstica(i, ladja, marsovec):
    marsovec = f"{marsovec[0]:10.4f} {marsovec[1]:10.4f}"
    if i is None:
        return f"{' ':4}  {' ':10} {' ':10}    {marsovec}"
    else:
        return f"{i:4}: {ladja[0]:10.4f} {ladja[1]:10.4f} -> {marsovec}"
```

ali kaj podobnega. Nemara je tako res bolj prav.

navodila

Tule je bila edina znanost, kako se izogniti izpisom številk v neprvih vrsticah. Takole: imeli bomo dve zanki.

- Zunanja gre čez naše ladje in ta potrebuje številko; najlepše jo bo dobiti kar z `enumerate`. Kot argument bomo dodali `start=1`. Če tega trika ne poznamo, bo pač štel od `0` in bomo kasneje, v klicu funkcije vrstica, prištevali `1`.
- Notranja zanka bo tekla čez marsovske ladje. Tu moramo pri prvi dodati številko, pri ostalih pa namesto številke podamo `None`. Ali smo pri prvi ladji, najlažje vemo tako, da "enumeriramo" tudi marsovce. Kadar je števec različen od `0`, podamo `None`, kadar je enak `0`, pa podamo števec iz prve zanke. Recimo tako.

```
def navodila(razpored, ime_datoteke):
    outf = open(ime_datoteke, "w")
    for i, (ladja, marsovci) in enumerate(razpored.items(), start=1):
        for j, marsovec in enumerate(marsovci):
            outf.write(vrstica(None if j else i, ladja, marsovec) + "\n")
```

Izraz `None if j else i` bo vrnil `None`, kadar je `j` resničen (torej različen od `0`) in `i`, kadar je neresničen. Če takšnih izrazov ne znamo pisati, bo čisto lepo deloval tudi običajni `if`,

Ne bojte se pokvariti `i`. Zunanja zanka bo normalno delovala naprej; `enumerate` zna šteti tudi če vmes prirejamo imenu, v katerega je zanka vpisala tekočo vrednost, kakšne druge stvari.

Če se ne znajdemo z enumeriranjem marsovcev, pa prvega marsovca izpišemo pred notranjo zanko, v notranji zanki pa izpisujemo ostale.

```
def navodila(razpored, ime_datoteke):
    outf = open(ime_datoteke, "w")
    for i, (ladja, marsovci) in enumerate(razpored.items(), start=1):
        outf.write(vrstica(i, ladja, marsovci[0]) + "\n")
```

```
    for marsovec in marsovci[1:]:
        outf.write(vrstica(None, ladja, marsovec) + "\n")
```